AFIT/GCS/ENS/99M-1

A JAVA BASED HUMAN COMPUTER INTERFACE FOR A UAV DECISION SUPPORT TOOL USING CONFORMAL MAPPING

Thesis

Randy A. Flood, First Lieutenant, USAF
AFIT/GCS/ENS/99M

19990409 041

AFIT/GCS/ENS/99M-1

A JAVA BASED HUMAN COMPUTER INTERFACE FOR A UAV DECISION SUPPORT TOOL USING CONFORMAL MAPPING

THESIS

Presented to the Faculty of the Graduate School of Engineering

Of the Air Force Institute of Technology

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Randy A. Flood, B.S.

First Lieutenant, USAF

March 1999

THESIS APPROVAL

NAME: Randy A. Flood, 1st Lieutenant, USAF **CLASS:** GCS-99M

THESIS TITLE: A JAVA BASED HUMAN COMPUTER INTERFACE FOR A UAV DECISION SUPPORT TOOL USING CONFORMAL MAPPING

DEFENSE DATE: 9 March 1999

COMMITTEE:	**NAME/TITLE/DEPARTMENT**	**SIGNATURE**
Advisor	T. Glenn Bailey, Lieutenant Colonel, USAF Assistant Professor of Operations Research and Deputy Head Department of Operational Sciences Air Force Institute of Technology	[illegible]
Reader	Gregg H. Gunsch, Lieutenant Colonel, USAF Assistant Professor of Computer Engineering and Deputy Head Department of Electrical and Computer Engineering Air Force Institute of Technology	[illegible]
Reader	Timothy M. Jacobs, Major, USAF Assistant Professor of Computer Science Department of Electrical and Computer Engineering Air Force Institute of Technology	[illegible]

Acknowledgements

I'd like to thank my sponsor, the UAV Battlelab, my advisor Lieutenant Colonel Thomas Bailey, my committee members Lieutenant Colonel Gregg H. Gunsch, and Major Tim Jacobs. I'd especially like to thank the 11th Reconnaissance Squadron. Without their support, none of this would have been possible.

Table Of Contents

List of Figures

List of Tables

AFIT/GCS/ENS/99M-1

Abstract

This paper describes the development of the Human Computer Interface (HCI) for a Decision Support System for routing Unmanned Aerial Vehicles (UAVs). This problem is a multi-vehicle routing problem with time-windows. Because of the unique nature of UAVs, a tool is needed to support dynamic re-routing. We solve the problem in two ways. First, we create a UAV Decision Support Tool (UAV DST) that uses a set of Java software objects to display maps and convert between latitude-longitude coordinates and x-y coordinates. Secondly, this library provides the ability for the user to dynamically re-optimize large UAV routing problems through a simple graphical interface. The library is built on top of a Java implementation of the tabu search algorithm written by O'Rourke (1999). This library provides the basis for future simulation and analysis of the Kenney Battlelab Initiatives by providing the interface to routing decision support and simulation modules.

A JAVA BASED HUMAN COMPUTER INTERFACE FOR A UAV DECISION SUPPORT TOOL USING CONFORMAL MAPPING

I. Introduction

The UAV Battlelab sponsored this research to investigate ways to more effectively use Uninhabited Aerial Vehicles (UAVs) to meet Air Force objectives. Specifically, we look at the Predator. The Predator is a slow UAV, with a long endurance that is typically used for reconnaissance operations. It broadcasts live video for rapid analysis. A typical Predator mission might have 50-100 targets, versus one or two targets for a fighter mission. While a fighter mission might last 2-3 hours, a Predator mission lasts 24-36 hours. Unlike targets for fighter missions, Predator targets have short time-windows, and unpredictable loiter times.

Currently, the 11th Reconnaissance Squadron, in Indian Springs Nevada, plans and executes missions using the Predator UAV. Operators begin with a list of targets, with associated time-windows. Using a Ground Control Station (GCS), operators manually enter route points by clicking on a map using subjective criteria for the ordering of the route points. The operator picks a route that looks good. The operator then performs a terrain clearance check, which ensures the Predator doesn't fly into a mountain; and, a line-of-sight check, which ensures that the Predator doesn't fly behind any mountains. This leaves them with an initial route for their mission.

For a number of reasons that will be explored later, the Predator operators must often re-plan their routes dynamically. Currently, there is no tool to help the operator re-plan the route dynamically. Each time the route is re-planned the operator must pick the order that they plan to visit the targets. If they make a sub-optimal decision, then they will not be able to image all of the planned targets.

We create a UAV Decision Support Tool (UAV DST) that helps the operators make this decision. O'Rourke (1999) creates a Java implementation of the tabu search algorithm for UAV routing, while Walston (1999) provides a discrete event simulation of UAV characteristics.

II. Implementing The UAV Decision Support Tool

2-1. Introduction And Literature Review

The Air Force is researching Unmanned Aerial Vehicles (UAVs) for missions involving a high risk of losing an aircraft, requiring a low cost platform, or requiring long endurance. One such application is the Suppression of Enemy Air Defenses (SEAD) mission; since enemy air defenses are designed to destroy aircraft, UAVs can expect to be targeted. In addition to using UAVs in new ways, there is also ongoing research in the areas of vehicle improvements. Both of these efforts can be significantly enhanced through the use of virtual prototyping.

The Air Force organization chartered to evaluate this area, and the sponsor of this research, is the UAV Battlelab. The mission of the UAV Battlelab is "...to rapidly identify and demonstrate the military worth of innovative concepts which exploit the unique characteristics of UAVs to advance Air Force combat capability." (Theisen 1999)

The UAV Battlelab accomplishes this mission by answering questions in the form of Battlelab Initiatives. According to the UAV Battlelab:

> A Battlelab Initiative is a concept or idea that may enhance the way the Air Force applies global air and space power. Ideas may be driven by combat experience, technology, or a desire to employ forces more effectively or efficiently. The Battlelab takes these ideas and concepts, and attempts to prove their value/worth to the Air Force. Initiatives are classified in terms of their scope as either Mitchell Class Battlelab Initiatives or Kenney Class Battlelab Initiatives (Theisen 1999).

This research is part of several Kenney Battlelab Initiatives (KBIs).

Kenney Battlelab Initiatives (KBIs) are for innovative, straight forward, and lower cost concepts. This category is named for Lt Gen George Kenney who adapted existing weapons and tactics to help turn the tide in the Pacific during the early days of World War II. Some examples of his work are parafrag bombs (hanging parachutes on small bombs to allow for bombing against aircraft in revetments), skip bombing against ships (adopted medium bombers to drop bombs at low altitude and placed cannons in the nose for more effective strafing), and what became called "Kenney Cocktails" (phosphorus bombs that exploded in the air sending out hot phosphorus to burn enemy aircraft in revetments). KBIS will be pursued under the sponsoring operating command's direction (Theisen 1999).

One KBI of interest is concerned with using UAVs for the SEAD mission. The 11th Reconnaissance Squadron tests the operational effectiveness of the Predator UAV. Currently, an operator from that squadron enters the route points that the UAV will fly. (There are up to 180 route points in a typical mission.) A collaborative research effort provides a decision support system for routing UAVs that requires a user interface for effective implementation.

The airmen who operationally route UAVs manually design target sequences by hand, and do not have the computer support to visually experiment and test their decisions with a routing decision support tool. This research provides such a capability by plotting target locations, then using an AutoRoute feature to calculate near-optimal routes with minimal travel time. A second collaborative research effort creates a discrete event simulation to support virtual prototyping of UAVs to evaluate capability improvements. For example, a user can double the speed of the UAV and determine the effect that has on the number of covered targets.

A significant challenge is accurately getting coordinate inputs from a map. While the Earth has a curved surface, maps are flat; hence they distort the size and shape of the landmasses. Software that displays maps need routines that convert between latitude-longitude coordinates to x-y coordinates. Previous research (Taylor 1997) has created routines in C and FORTRAN to do this for meteorological software. The literature provides routines to do these transformations (Taylor 1997, Allison 1995, Bortoluzzi and Ligi 1986). Some of the software routines (e.g. W3LIB) require every single map parameter with every function call to convert coordinates. Others maintain global data structures with this information that prevent working with more than one map at a time. (e.g. EZMAP). Taylor created routines that use initialization routines to fill in C structures, thus allowing a library to support more than one map at a time.

This research creates a library of objects in Java to display maps, and convert the coordinates from x-y to Latitude-Longitude. Java is an object oriented programming language created by Sun Microsystems for embedded applications. Its main advantage over traditional languages is that it's portable across many platforms and operating systems. Java also allows the creation of applets, which can be executed from Web pages by major browsers such as Netscape and Microsoft Internet Explorer. Our library provides the Human Computer Interface (HCI) for discrete event simulations of UAVs and routing algorithms to support the modeling and support of KBIs.

The literature provides much research into algortihms for the multi-vehicle routing problem. Bertsimas and Simchi-Levi (1996) gives a summary of algorithms for the vehicle routing problem. This includes best and worst case analysis for many algorithms. Gendreu et al. (1996) describes the use of tabu search on a class of the vehicle routing problem where there are random demands. They find an optimal solution 89.45% of the time. Ryan et al. (1999) describe using the tabu search algorithm for the UAV routing problem in Modsim. O'Rourke (1999) applies the tabu search algortihm to the UAV routing problem in Java.

The literature provides good reasons for building a graphical display for this problem. Crossland et al. (1995) examines whether the addition of Geographic Information Systems (GIS) to decision support systems affects the performance of individuals on spatial decision problems. The study found "unequivocal evidence" that the use of GIS increased the accuracy of decision-makers, as well as reduced the decision time. Keenan (1998) notes that while standard GIS software can be useful to a broad range of routing problems, a general purpose GIS will not be suitable for complex multi-vehicle routing problems. Keenan also notes that a skilled user can dramatically improve the routes generated by a heuristic routing function through skilled manipulation. Basnet (1996) create a Decision Support System (DSS) for a particular vehicle routing problem that arises in the New Zealand dairy industry. They create a user interface in Pascal that runs as a DOS program.

How to create user interfaces for DSSs is another focus of research. Jones (1991) gives a taxonomy of the types of user interface development breaking it down into: subroutine libraries, draw-it yourself, hypermedia toolkits, object-oriented, text languages, network, by example, syntax-directed editors, and constraint-based. Jones argues that user interfaces are an important and neglected part of DSSs. Angehrn (1990,

1991) creates a flexible system for graphically creating DSSs called Tolomeo. The basic idea is to let users specify specific examples of the problem they face, and some of the kinds of solutions they are looking for. The system then forms a hypothesis about the formal nature of the problem, and selects mathematical methods for solving it. Finally, it suggests new solutions to the user. Holsapple et al. (1991) describes a complicated framework for developing user interfaces for DSSs, dividing the effort into interface, event and functionality development. They create languages for describing customized decision support system interfaces.

The literature, then, contains several distinct focuses. Some research concentrates on algorithms for the multi-vehicle routing problem. Other research examines the benefits of integrating GIS with DSSs. Finally, some research concentrates on frameworks for creating user interfaces for DSSs.

This chapter is organized in the following manner. Section 2-2 explains the operational background for this problem, including the routing algorithm, and the unique characteristics of the UAV environment. Section 2-3 explains the design of the user interface, including the algorithms used for conformal mapping, as well as the integration of locked subroutes and threats with the routing algorithm. Section 2-4 explains the operational contribution of this research. Section 2-5 describes significant implementation details, and Section 2-6 concludes this thesis with a summary and suggestions for further research.

2-2. Operational Background

The UAV routing problem, or UAVP, is in the most general sense a special case of the Traveling Salesman Problem. Ryan et al. (1999) explain how the UAVP problem fits into Carlton's taxonomy of general vehicle routing problems (GVRP). Since UAVP is a homogeneous, multiple-vehicle, single-depot, traveling salesman problem with route-length constraints, and time windows, it is characterized as a [MVH, SD, TSP, RL, TW]. Ryan et al. (1999) further note that since GVRP belongs to the class of NP-complete problems, a heuristic method should be used to find near optimal solutions. Ryan et al.'s (1999) solution to the problem was to develop a MODSIM program using reactive tabu search on the TSP problem with time windows.

O'Rourke (1999) extends Ryan et al.'s (1999) research, and creates a Java program that performs reactive tabu search to solve the UAVP. However, there are several unique aspects of the UAV environment that are not directly handled by O'Rourke's routines. First, there is the notion of threats; e.g. a Surface to Air Missile (SAM) site may render certain route segments dangerous to fly on. Another unique aspect of the environment is the concept of *locked sub-routes*. Locked sub-routes are route segments that the user tells the algorithm to retain during its searching. This is essential because there are often certain air corridors that must be flown when entering and leaving controlled airspace, or certain route segments the operator knows *a priori* must be part of the solution.

The Predator system consists of the Predator aircraft, the ground control station (GCS), data links, sensor payloads, ground support equipment, and trained personnel.

The GCS is a trailer that contains a mission planning station, a data exploitation station, an air vehicle operator station and a payload station. The Predator is remotely piloted from the GCS. The Predator must take off and land near the GCS since there are delays in response time due to the line of sight communications. In theory, a UAV could take off from one GCS, and be passed off to another mid-flight. However, the current doctrine prevents this from occurring.

Table 1 shows a notional list of targets for the Predator. Figure 1 shows a sample plot for a Predator mission.

Table 1. Notional Predator Target List (Ryan 1999)

Notional Bosnia Scenario															
	R									First Visit				Second Visit	
	O	#1	#2	Latitude			Longitude			Early	Late	Service Time		Early	Late
Target Name	Z	ID	ID	Deg	Min	Sec	Deg	Min	Sec	Arrival	Arrival	Ranges (min)		Arrival	Arrival
Taszar Hungary, Depot				46	24	0	17	54	0						
Corridor, Szulok Hungary				46	3	45	17	32	44						
Corridor, Srbac Bosnia				45	24	0	17	30	0						
Dumdvga	1	1	32	44	58	29	16	50	34	1015	1500	30	180	1900	2300
Mastye	1	2	33	44	58	46	16	38	56	1015	1500	30	180	1900	2300
Garred AAA Site	1	3	34	44	58	4	16	39	31	1015	1500	2	15	1900	2300
Tharmet Heavy Weapons Depot	1	4	35	44	58	33	16	39	18	1015	1500	2	30	1900	2300
Tharmet Heavy Weapons Depot	1	5	36	44	58	39	16	39	41	1015	1500	2	30	1900	2300
Tharmet Heavy Weapons Depot	1	6	37	44	58	59	16	39	28	1015	1500	2	30	1900	2300
Serdona Communications Site	1	7	38	44	59	2	16	39	56	1015	1500	2	30	1900	2300
Serdona Communications Site	1	8	39	44	59	11	16	40	19	1015	1500	2	30	1900	2300
Serdona Communications Site	1	9	40	44	59	15	16	39	20	1015	1500	2	30	1900	2300
Suspected Weapons Storage	1	10	41	44	59	9	16	39	10	1015	1500	2	30	1900	2300
Suspected Weapons Storage	1	11	42	44	54	52	16	34	47	1015	1500	2	30	1900	2300
Suspected Weapons Storage	1	12	43	44	51	49	16	41	37	1015	1500	2	30	1900	2300
Suspected Weapons Storage	1	13	44	44	0	7	16	34	47	1015	1500	2	30	1900	2300
Suspected Weapons Storage	1	14	45	44	59	9	16	49	17	1015	1500	2	30	1900	2300
Suspected Weapons Storage	1	15	46	44	57	41	16	39	35	1015	1500	2	30	1900	2300
Air Defense, SAM, Probable SA-2	1	16	47	44	57	23	16	51	45	1015	1500	2	30	1900	2300
Air Defense, SAM, Probable SA-2	1	17	48	44	57	45	16	49	28	1015	1500	2	30	1900	2300
Air Defense, SAM, Probable SA-2	1	18	49	44	55	57	16	43	52	1015	1500	2	30	1900	2300
Air Defense, SAM Site Radar	1	19	50	44	57	47	16	39	54	1015	1500	2	30	1900	2300
Dromada HQ Site	1	20	51	45	0	7	16	53	49	1015	1500	30	120	1900	2300
Dromada Warehouse	1	21	52	44	53	31	16	54	12	1015	1500	2	60	1900	2300
Omanski Barracks	2	22		44	45	34	17	10	34	1015	1715	5	120		
Omanski Barracks	2	23		44	48	19	17	12	14	1015	1715	5	120		
Omanski Barracks	2	24		44	51	2	17	13	24	1015	1715	5	120		
Bolstavec Tank Rally Point	2	25		44	50	51	17	14	39	1015	1715	2	30		
Bolstavec Tank Rally Point	2	26		44	56	17	17	17	41	1015	1715	2	30		
Krajachastane Storage Bunker	2	27		44	55	51	17	17	51	1015	1715	2	30		
Krajachastane Storage Bunker	2	28		44	56	7	17	18	23	1015	1715	2	30		
Goldprtunity Road	3	29		44	28	13	17	1	18	1015	1830	20	40		
Goldprtunity Road	3	30		44	27	29	17	1	46	1015	1830	20	40		
Goldprtunity Road	3	31		44	27	10	17	2	24	1015	1830	20	40		

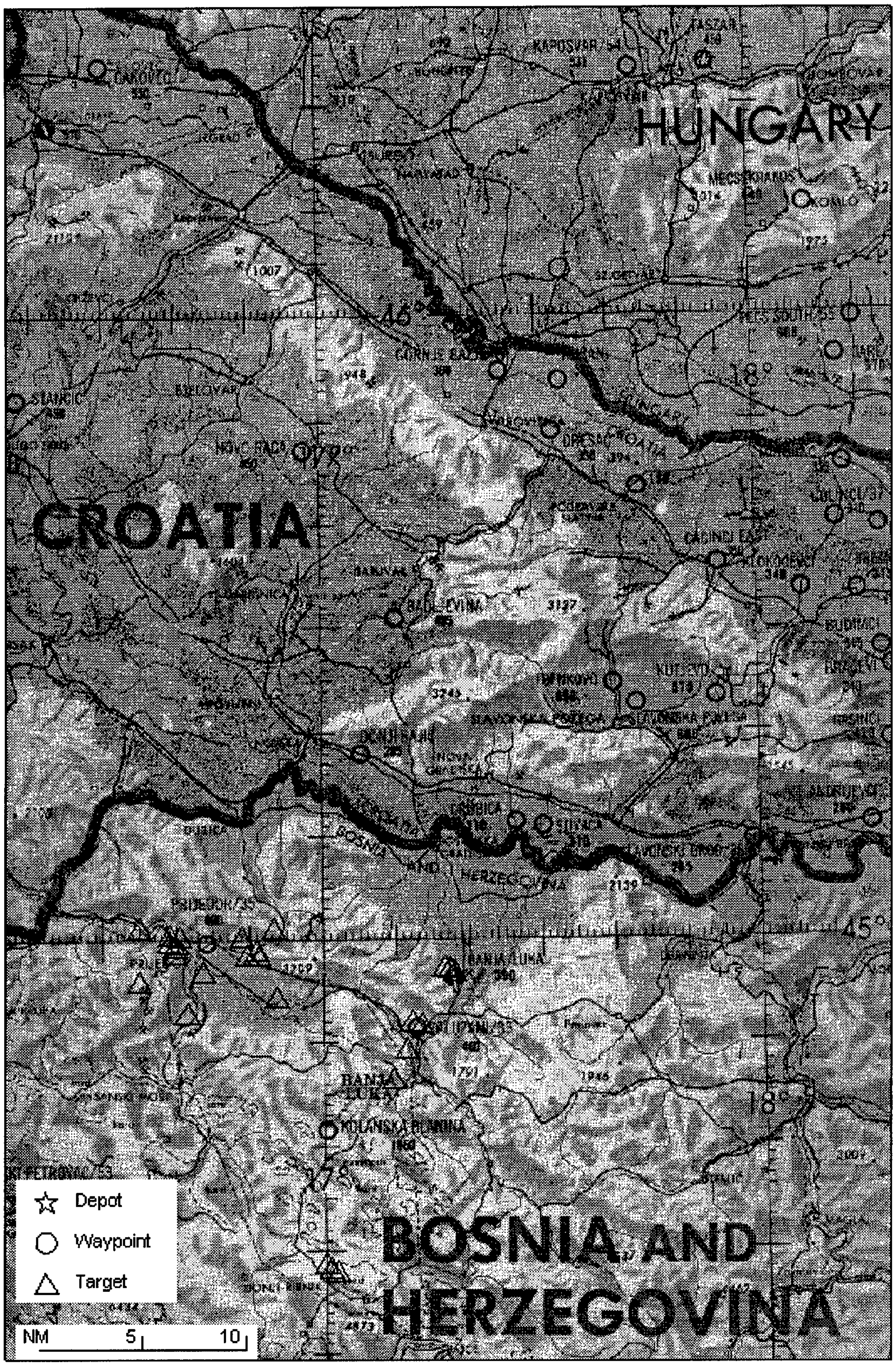


Figure 1. Sample Plot (O'Rourke 1999)

Table 2 shows the performance characteristics of the Predator.

Table 2. Predator Performance Characteristics (Sisson 1997)

Predator Performance Characteristics	
Maximum altitude	25,000 ft
Maximum endurance	40+ hours
True Air Speed	60-129 knots
Cruise Speed	70 knots
Radius	500 Nm
Sensors	SAR, EO, IR
Thrust	85 Hp
Length	26.7 ft
Width	3.7 ft
Navigation System	GPS, INS
Survivability Measures	None
Payload	450 lbs

The Predator has several interesting characteristics. First, it flies at extremely slow speeds. In fact, the Predator often flies too slow to be picked up on radar, and it is sometimes slower than the wind. Predators have been known to have a negative groundspeed. Second, the Predator sends back live video to intelligence. The Predator contains electro-optical infra-red (EO)/(IR) sensors, which consist of an infra-red camera for night missions, and two video cameras for use during the day. The Predator uses these sensors to send live video back to the GCS. Since the video is live, and easily understandable, this prompts a lot of requests to reroute the aircraft during flight to get a better look at things. Third, the Predator is very sensitive to bad weather. It does not fly well in the rain, because the water seeps through its wings and damages its electronics. (The camera for the Predator is much more expensive than the airframe!) Also, if ice forms on the Predator's wings, it becomes aerodynamically unstable. Fourth, the

Predator is entirely unclassified. This means that there are far fewer restrictions on where it can fly than a U2.

All of these characteristics force the Predator operators to re-plan their routes frequently. During a typical mission, the aircraft is often diverted from its original route to cover unanticipated targets. Likewise, since it has trouble flying against the wind, and since it does not perform well in the rain, the operator often needs to re-plan the route dynamically to account for weather. Each time the operator re-plans the route, he or she must make a decision about what order to visit the targets in. If the operator makes a poor decision, there will not be enough time to cover all of the targets.

Currently, mission planning is done using the GCS. Operators take a list of targets, and enter their coordinates into the GCS to plan a route. Usually, this is done by clicking on a map, though the capability to enter latitude/longitude coordinates is also availiable. The GCS performs a terrain analysis, which ensures the route does not go through a mountain, as well as a communications profile, which ensures that line-of-sight communications is maintained at all times. However, the GCS does not provide any insight into what order to visit the targets in.

2-3. Interface Considerations

This research creates an application that demonstrates an automatic route-planning feature (AutoRoute) using the tabu search algorithm. A separate research effort by O'Rourke (1999) implements the tabu search algorithm in Java. Figure 2 shows the Uninhabited Aerial Vehicle Decision Support System (UAV DST) application.

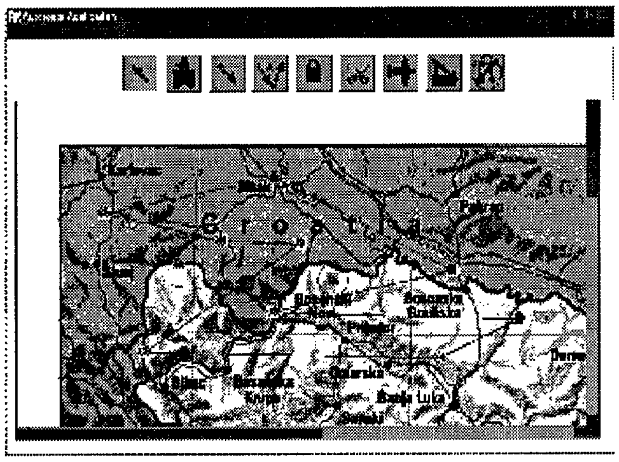


Figure 2. The UAV DST Application

Figure 3 shows the name of each of the buttons. We present a detailed description functional use and capabilities of features listed in Figure 3.

- Selection
- Ground Control Station
- Add Target
- AutoRoute
- Lock
- Cut
- Aircraft Characteristics
- Add Threat
- Add No-Fly Zone

Figure 3. Descriptions of Buttons

2-3-1 Selection.

The *Selection* tool selects objects. Using the *Selection* tool, clicking on a target, and then releasing the mouse button, will select that target, and display the *Target Characteristics Dialog Box.* After selecting a target, you may click on it and drag it across the map to move it. When you move a target, the route follows. Moving a threat or a node in a no-fly zone works the same way. Simply select it, then click on it and drag it across the map. Selecting a target, without releasing the mouse button, and then dragging it on top of another target will create a locked route segment from the first target to the second one. This tool will be used whenever you need to move something on the map, or manually adjust the route.

2-3-2 Ground Control Station.

The *Ground Control Station* (GCS) tool inserts a ground control station on the map. Using the *Ground Control Station* tool, clicking on the map, and releasing the mouse button will move the GCS to the place where you clicked. The GCS acts as the depot to the routing algorithm, and thus is the point where all UAVs take-off and land. For this application, there is only one GCS. This tool is only used when you want to move the GCS, which is infrequently.

2-3-3 Add Target

The *Add Target* tool adds targets to the map. Using the *Add Target* tool, clicking on the map, and releasing the mouse button will add a target to the map at the point where you clicked. To move a target on the map, you must select it, and drag it across the map using the *Selection* tool. To edit the characteristics of a target, you must select it using the *Selection* tool. Targets act as the customer nodes to the routing algorithm. The *Add Target* tool is used whenever you need to add a new target to the map, which is very frequently.

2-3-4 **AutoRoute.**

The *AutoRoute* button begins calculating a near-optimal route. Clicking the *AutoRoute* tool will begin calculating a near-optimal route using 3,500 iterations of the tabu search algorithm. The cursor changes to an hourglass indicating that the system is busy. When the new route is displayed, and the cursor changes back to the arrow cursor, then the AutoRoute calculation is complete. You should use the *AutoRoute* button whenever you add or remove one or more targets, threats, or no-fly zones to the map, or move anything on the map. This is the key feature of this application. It is intended to be used frequently.

2-3-5 Lock.

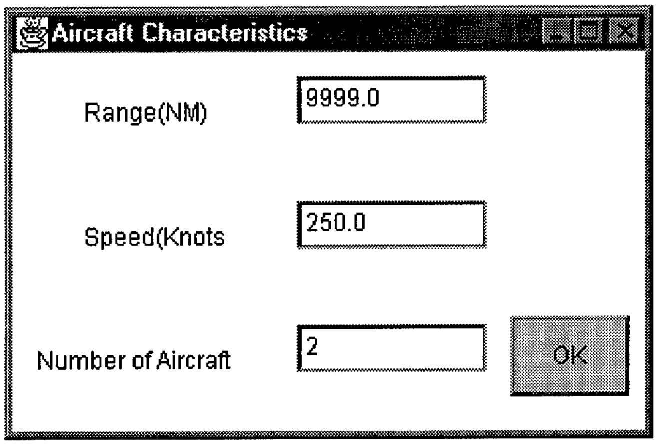
The *Lock* tool allows the user to lock route segments, so that they will not be changed by the AutoRoute feature. Using the *Lock* tool, clicking on a target locks the route segment immediately after that target. Clicking the same target again using the *Lock* tool unlocks the route segment. You would use this tool to lock any part of the route that you don't want the AutoRoute feature to change. For example, you can use the lock tool to ensure that the AutoRoute feature will not change the part of the route that flies through controlled airspace. Also, if you have a target that you know you must visit next, you can lock that portion of the route. This feature is designed to be used somewhat frequently.

2-3-6 Cut.

The *Cut* tool is used to remove targets, threats, and no-fly zones from the map. Using the cut tool, clicking on a feature on the map removes it. Alternatively, selecting a feature and then clicking on the cut tool also deletes that feature. Deleting the last node in a no-fly zone deletes it. The *Cut* tool is used whenever you want to delete a target, threat, or node in a no-fly zone from the map.

2-3-7. Aircraft Characteristics.

The *Aircraft Characteristics* button displays *the Aircraft Characteristics Dialog Box* (Figure 4). There are three parameters that can be modified. Parameters can be changed by clicking on the field for that parameter, then entering a new value, then clicking the *OK* button.



Range(NM) 9999.0

Speed(Knots 250.0

Number of Aircraft 2

Figure 4. Aircraft Characteristics Dialog Box

2-3-8. Add Threat.

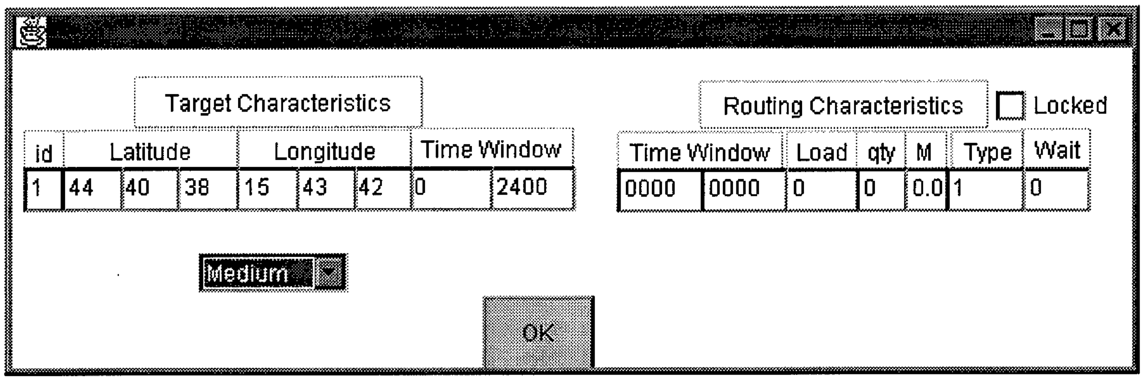
The *Add Threat* tool is used to add threats to the map. Using the *Add Threat* tool, clicking on the map adds a threat at the point where you clicked. To move threats, use the *Selection* tool to drag them across the map. To edit the properties of threats, select the threat using the *Selection* tool, then edit the desired properties in the *Threat Characteristics Dialog Box.* This tool will be used whenever you need to add a threat to the map. Due to the mostly static nature of threats, this tool will be used infrequently.

2-3-9. Add No-Fly Zone

The *Add No-Fly Zone* tool is used to add no-fly zones. Using the *Add No-Fly Zone* tool, clicking the corners of a polygon creates a new no-fly zone. To add new points to an existing no-fly zone, first, select it, using the *Selection* tool, then, after clicking on the *Add No-Fly Zone* tool, clicking on the map will add points to the selected no-fly zone. This tool is used whenever you need to add another no-fly zone to the map.

2-3-10. Target Characteristics Dialog Box

When a user clicks on a target using the selection tool, the dialog box shown in Figure 5 is displayed. As the user drags the target on the map, the latitude and longitude coordinates are updated in the dialog box. This allows the user to accurately position the target on the map. Alternatively, the user can enter the latitude longitude coordinates in the dialog box, and press the OK button.



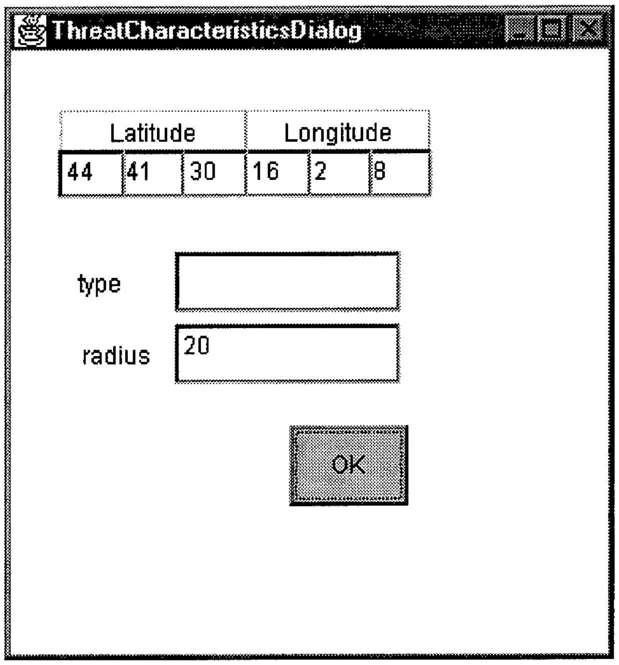
id	Latitude			Longitude			Time Window	
1	44	40	38	15	43	42	0	2400

Time Window		Load	qty	M	Type	Wait
0000	0000	0	0	0.0	1	0

Figure 5. Target Characteristics Dialog Box

2-3-11. Threat Characteristics Dialog Box.

If the user clicks on a threat using the selection tool, then the Threat Characteristics dialog box is displayed (see Figure 6). Once again, as the user drags the threat across the map, the latitude and longitude are dynamically updated.



	Latitude			Longitude	
44	41	30	16	2	8

type

radius 20

Figure 6. Threat Characteristics Dialog Box

2-3-12. Aircraft Characteristics Dialog Box.

If the user clicks on the *Aircraft Characteristics* button, or selects *aircraft characteristics* from the view menu, the *Aircraft Characteristics Dialog Box* is displayed, (Figure 3).

2-4. GUI/Tabu Interface

The tabu search algorithm inputs an array of N+v+1 nodes numbered 1..N+v+1, with associated early arrive times e_i, late arrival time l_i, and wait-time w; a number of vehicle Nodes v; a number of customer (i.e. target) nodes N; a (N+v+1 by N+v+1) time/distance matrix D; and outputs an ordered list of a near-optimal route. The routing algorithm assumes that the first node is a vehicle node, and that the last node is the place for the aircraft to stop upon completing its tour (which in most cases is the same as the first node.)

There are several challenges associated with using this tabu search algorithm in the context of this application. The first challenge is the notion of *locked sub-routes*. Locked sub-routes are route segments that the user tells the algorithm to retain during its search. This is essential because certain air corridors must often be flown when entering and leaving controlled airspace. Additionally, the user may be required to divert the aircraft to survey an unanticipated target, and does not want the algorithm to change one or more portions of the route that are already flight planned or profiled for terrain clearance and communication.

Initially, all route segments are eligible for inclusion in the suggested route. The combined use of the tabu search algorithm and locked subroutes poses a unique implementation challenge. One method of accomplishing this is to divide up the nodes such that the tabu search algorithm only considers a subset of the route at a time. Under this approach, the tabu search would consider a route that includes the first node in the locked sub-route, but excludes other nodes in the locked sub-route. Then, it would plan a route starting with the last node in the locked sub-route, using only the remaining nodes. This technique concludes by piecing together these sub-routes. However, this approach while finding local optimums, may not find a global optimum. Also, it is difficult to determine how to group the nodes in the first part of the locked sub-route.

Instead of a direct representation of the nodes into the routing algorithm, all of the nodes in a locked sub-route are grouped into a single supernode. For example, if nodes $N_i..N_j$ form a locked subroute, a single supernode M_i, represents them to the routing algorithm, with a wait-time equal to the sum of the component wait times in $N_i..N_j$. In the time/distance matrix, the distance from any node N_k to M_i is the distance from N_k to N_i ; however, the distance from M_i to N_k, is equal to the sum of the distances from $N_i..N_j$ plus the distance from N_j to N_k.

After the tabu search returns a route, the supernodes are translated back to the locked subroute node segments through replacement. This creates a new route, that contains no supernodes, yet retains the desired locked sub-routes.

As discussed earlier, another difficulty with using the tabu search algorithm in this domain is the concept of threats. The UAV DST models threats using a latitude/longitude coordinate and a radius. When building the time/distance matrix, any route segment which intersects the circle around a threat is given an extremely large penalty in the time/distance matrix. By making any solution containing that route segment infeasible, the routing algorithm will prefer routes that avoid threats.

Although in many cases the output of the AutoRoute feature will be accepted, the user may need to manually adjust the route. We allow the user to drag one target to overlay another in a way that creates a route from one node to the next (Figure 7).

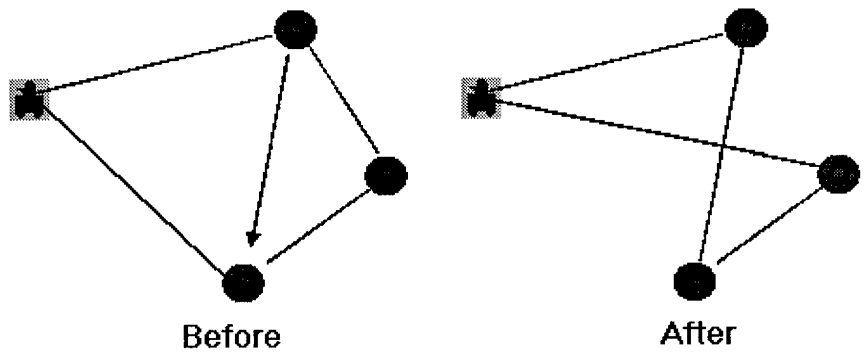


Figure 7. The user drags the top target onto the bottom one

2-5. Conformal Mapping

Another significant challenge is inputting coordinates from a map. In order to do this, a conformal map object is developed. Map projections are systematic ways of transferring the 3-dimensional geometry of the Earth's surface on to a 2-dimensional surface (such as a piece of paper or a computer screen.) This can be viewed in terms of shadow casting, such as a light inside a globe casting shadows on a specially shaped paper near the globe. The shape of the paper used determines the type of projection; for example, a paper shaped as a cylinder gives a cylindrical projection, a paper shaped like a cone provides a conic projection, while a flat or planar sheet of paper provides a zenithal or azimuthal projection(Hill 1989). Table 3 lists some features of the most common projections.

Table 3. Some features of common projections (taxonomy based on Dana 1995)

Projection	Family	Key Features	Common Usage	Implementation
Mercator	Cylindrical	Straight meridians and parallels that intersect at right angles. Scale is true at the equator or at two standard parallels equidistant from the equator.	Marine navigation	(Taylor 1997) - Fortran & C (Allison 1995)- Turbo Pascal Bortoluzzi- Fortran
Lambert Conformal Conic	Conic	Area and shape are distorted away from standard parallels. Directions are true in limited areas.	Maps of North America.	Allison- Turbo Pascal (Bortoluzzi 1986)- Fortran
Polyconic	Conic	Latitude Lines are arcs from circles. The central Meridian and the equator of the projection are straight lines. The projection is free of distortion only along the central meridian. (Allsion 1995)	Large scale mapping of the United States	(Allison 1995)- Turbo Pascal
Albers Equal Area Conic	Conic	Lines of latitude are unequally spaced arcs of concentric circles more closely spaced at the north and south edges of the projection. Preserves the area dimensions of equal latitude-longitude extents. Lines of longitude are equally spaced radii of the same circles, and cut lines of longitude at right angles. There is no distortion of scale or geometry along the two standard parallels.(Allison 1995)	Equal-area maps with large east-Ist expanse.	(Allison 1995)- Turbo Pascal
Transverse Mercator	Cylindrical	The central Meridian and Equator of the projection are straight lines. Scale is true along the central meridian or along two straight lines equidistant and parallel to the meridian. Scale becomes infinite 90 degrees from the central meridian.	Quadrangle maps with scale ranging from 1:24,000 to 1:250,000	(Allison 1995)- - Turbo Pascal
Universal Transverse Mercator	Cylindrical	Defines horizontal positions by dividing the surface of the Earth into 6 degree zones, each mapped by the Transverse Mercator projection with a central meridian in the center of the zone.		(Allison 1995)- - Turbo Pascal (Bortoluzzi 1986)- Fortran
Polar Stereographic	Azimuthal	Directions are true from the center point and scale increases away from the center point as does distortion in area and shape.	Navigation in polar regions	(Taylor 1997) - Fortran & C (Bortoluzzi 1986)- Fortran

This UAV DST implements a Mercator projection. According to Taylor latitude and longitude to x-y conversion is defined as

$$X = X_0 + \frac{a}{G_0}(C_1\xi + C_2\eta)$$

$$Y = Y_0 + \frac{a}{G_0}(C_1\xi - C_2\eta)$$

where ξ and η are the latitude and longitude coordinates of the point, a is the radius of the Earth, G_0 is the gridsize at the equator, and $C1$ and $C2$ are constants.

Converting from x-y coordinates to latitude–longitude uses the following equations

$$\xi = \frac{G_0}{a}[\, c_1(x - x_0) - c_2(y - y_o)]$$

$$\eta = \frac{G_0}{a}[\, c_1(y - y_o) + c_2(x - x_o)] .$$

Supporting conformal mapping in Java requires the classes Xy and LatLong for storing x-y coordinates and latitude-longitude coordinates, respectively. The Xy class supports the following methods shown in Table 4.

Table 4. Class Xy

Method	Description
public Xy(int x, int y)	Constructor
public int getX()	Assessor function for the X coordinate
public int getY()	Assessor function for the Y coordinate

The methods for the LatLong class are given in Table 5.

Table 5. Class LatLong

Method	Description
public LatLong(double Lon, double Lat)	constructor for specifying LatLong coordinates doubles
public LatLong(int LongDegrees, int LongMinutes, int LongSeconds, int LatDegrees, int LatMinutes, int LatSeconds)	constructor for specifying LatLong coordinates Degrees, Minutes, and seconds
public final int getLongDegrees()	Assessor function for the Degrees Longitude
public final int getLatDegrees()	Assessor function for the Degrees Latitude
public final int getLatMinutes()	Assessor function for the Minutes Latitude
public final int getLongMinutes()	Assessor function for the Degrees Longitude
public final int getLongSeconds()	Assessor function for the Seconds Longitude
public final double getLat()	Assessor function for the Latitude as a double
public final double getLong()	Assessor function for the Longitude as a double
public final void setLat(double L)	Sets the Latitude as a double
public final void setLong(double L)	Sets the Longitude as a double
public final void setLatDegrees(int d)	Sets the Degrees of Latitude
public final void setLongDegrees(int d)	Sets the Degrees of Longitude
public final void setLatMinutes(int m)	Sets the Minutes of Latitude
public final void setLongMinutes(int m)	Sets the Minutes of Longitude
public final void setLatSeconds(int s)	Sets the Seconds of Latitude
public final void setLongSeconds(int s)	Sets the Seconds of Longitude
public void print()	Prints the Latitude and Longitude
public void printLat()	Prints the Latitude as a double
public void printLong()	Prints the Longitude as a double

In order to support conformal mapping, we create a ConformalMap Class in Java. The conformal map object initializes by passing in the x-y coordinates and the latitude-longitude coordinates of two known points. Table 6 shows the methods in ConformalMap.

Table 6. Class ConformalMap

Method	Description
public ConformalMap(Xy P1, LatLong L1, Xy P2, LatLong L2)	Constructor, which takes 2 X-y coordinates, along with their corresponding LatLong coordinates
Public LatLong Xy2LatLong(Xy P)	Converts Xy coordinates to LatLong coordinates
public Xy LatLong2Xy (LatLong P)	Converts LatLong coordinates to coordinates to Xy coordinates
public double getDistanceBetween (LatLong P1, LatLong P2)	Returns the great circle distance between 2 LatLong coordinates
public void print()	Prints all the variables in ConformalMap for debugging purposes
public double distanceBetween(Xy P1, Xy P2)	Returns the Cartesian distance between 2 Xy coordinates
boolean LineThoughThreat(Xy C, Xy P1, Xy P2, int R)	Determines if a line segment defined by 2 Xy points intersects a circle at C with radius R

The constructor for the ConformalMap class calculates the parameters for coordinate conversion as follows. Beginning with the constructor

public ConformalMap(Xy P1, LatLong L1, Xy P2, LatLong L2)

let x_a and y_a be the x and y coordinates of *P1* respectively. Let x_b and y_b be the x and y coordinates of *P2*. Let η_a be the longitude of *P1*, and ξ_a be the latitude of *P1*. Let η_b be the longitude of *P2*, and ξ_b be the latitude of *P2*. G_0 is the gridsize at the equator. d_x is the Cartesian distance between *P1* and *P2* in x-y coordinates. d_ξ is the Cartesian distance between *P1* and *P2* in latitude-longitude coordinates. C_1, and C_2 are constants. x_0 and y_0 are the longitude and latitude of the x-y coordinate (0,0).

Following Taylor (1997) the following calculations are performed:

$$d_x = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$$

$$d_\xi = \sqrt{(\xi_a - \xi_b)^2 + (\eta_a - \eta_b)^2}$$

$$G_0 = \frac{a d_x}{d_\xi}$$

$$c_1 = \frac{(x_a - x_b)(\xi_a - \xi_b) + (y_a - y_b)(\eta_a - \eta_b)}{d_X d_\xi}$$

$$C_2 = \frac{(x_a - x_b)(\eta_a - \eta_b) - (y_a - y_b)(\xi_a - \xi_b)}{d_X d_\xi}$$

$$x_0 = x_a - \frac{(c_1 \xi_a + c_2 \eta_a) d_X}{d_\xi}$$

$$y_0 = y_a - \frac{(c_1 \eta_a + c_2 \xi_a) d_X}{d_\xi}.$$

Once the ConformalMap object has been initialized, one can convert x-y coordinates into latitude-longitude coordinates by calling **public LatLong Xy2LatLong(Xy P).** Likewise, converting latitude-longitude coordinates into x-y coordinates is accomplished by calling **public Xy LatLong2Xy (LatLong P).**

The **boolean LineThoughThreat(Xy C, Xy P1, Xy P2, int R)** method determines if a line from P1 to P2 would intersect a circle centered at C with radius R. To understand how this works examine Figure 6 where

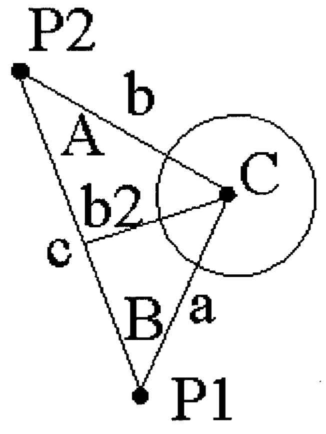


Figure 8. A circle with radius R at point C, and a line segment from P1 to P2

```
$a$ = distanceBetween(P1, C);
$b$ = distanceBetween(P2, C);
$c$ = distanceBetween(P1, P2);
$B = ACOS(-\frac{(b^2 - a^2 - c^2)}{2ac})$
$b2 = a(\text{SIN}(B))$
if ($b2<R$) return true;
else return false.
```

Using the law of Cosines:

$$b^2 = a^2 + c^2 - 2acCOS(B)$$

$$B = ACOS(-\frac{(b^2 - a^2 - c^2)}{2ac})$$

Now, the segment *b2* forms a right angle with the segment from *P1* to *P2*. Hence, $b2=a(\mathrm{SIN}(B))$. Now, if $b2 < R$, then the line intersects the circle.

2-6. Implementation Details

We develop the UAV DST application using the rapid prototyping model. We began by interviewing the manufacturers of several UAVs looking for a general understanding of their capabilities and unique characteristics. We then met with the 11th Reconnaissance Squadron to see how they used the Predator operationally, and what problems they have. Next, we discussed UAV issues with a staff officer in Air Combat Command long range planning.

At this point, we were able to develop the first version of the user interface. We chose Symantic Visual Café as our development platform, because it has powerful features for designing user interfaces. This allowed us to create our first prototype. It was extremely slow, and did not yet have the AutoRoute capability. We demonstrated this prototype to the 11th Reconnaissance Squadron. They gave us valuable feedback. They wanted the ability to resize the window, a zoom capability, and different priority nodes to be different colors.

We added the features they requested to the prototype, and integrated the tabu search algorithm developed by O'Rourke (1999). We returned to the 11th Reconnaissance Squadron, and demonstrated the second prototype. They were generally pleased. Some operators commented that it should be integrated into the mission planning software that intelligence officers use to plan missions. There was a general agreement that a routing algorithm should use priorities, but there was no consensus on exactly how priorities should be used.

In March 1999, we will to return to the 11th Reconnaissance with our final version of the UAV DST. We will deliver it to them on a laptop that they can take with them when they deploy.

2-7. Conclusion

We deliver a laptop containing the UAV DST application to the 11th Reconnaissance Squadron. Using our software, they will be able to generate routes more efficiently. Since their current software runs on a large UNIX workstation, it is difficult for users to plan routes away from the workstation. Using the laptop, users can experiment with different routes and then plug the best route into the workstation.

This research develops a ConformalMap class to handle conformal mapping in Java. Unlike previous routines, this software is object oriented and highly portable. A UAV DST is developed that demonstrates an automatic routing capability for UAVs. A number of interesting features are provided, including integrating locked subroutes and threats into the tabu search algorithm.

Future research needs to be done in several areas. First is the integration of the AutoRoute feature into the software already used operationally to create routes. Second, a separate research effort creates a discrete event simulation to model UAVs. The HCI libraries could be easily extended to provide a graphical user interface for the discrete event simulation. Finally, there are a couple of features of feasible routes that we did not model. For example, because of the need for line of sight communication some routes might not be feasible.

Appendix 1. Alphabetical Index Of Fields and Methods

A

AboutDialog(Frame, boolean). Constructor for class AboutDialog

Method AboutDialog is the constructor

AboutDialog(Frame, String, boolean). Constructor for class AboutDialog

Method AboutDialog is the constructor taking a string which acts as the title

actionPerformed(ActionEvent). Method in class myToolbarTestPanel

Method actionPerformed is the standard action callback

add(TimeWindow). Method in class NoFlyZoneContainer

Method add adds a NoFlyZone node (as a TimeWindow) to the current NoFlyZone

addNotify(). Method in class AboutDialog

Method addNotify is routine that is automatically generated by Symantic Visual Cafe

addNotify(). Method in class AircraftCharacteristicsF

Method addNotify is automaticallt generated by Symantic Visual Cafe

addNotify(). Method in class Frame1

Method addNotify is automatically generated by Symantic Visual Cafe

addNotify(). Method in class QuitDialog

Method addNotify is automatically generated by Symantic Visual Cafe

addNotify(). Method in class TargetCharacterisitcsWindow

Method addNotify is automatically generated by Symantic Visual Cafe

C

c(boolean). Method in class Frame1

Shows or hides the component depending on the boolean flag b.

compPens(NodeType[], int). Static method in class NodeType

compPens computes the exact vehicle Overload and time window penalties

compPens(NodeType[], int). Static method in class VrpPenType

compPens computes the exact vehicle Overload and time windoe penalties

ConformalMap(Xy, LatLong, Xy, LatLong). Constructor for class ConformalMap

Method ConformalMap is the constructor for the ConformalMap class

CoordType(). Constructor for class CoordType

CoordType(double, double). Constructor for class CoordType

copy(). Method in class NodeType

countVeh(NodeType[]). Static method in class NodeType

countVeh finds the number of vehicles being used in the current tour by counting the vehicle to demand transitions

countVehicles(NodeType[]). Static method in class TabuMod

countVeh calculates the number of vehicles used in the current tour by counting the number of vehicle (type 2) to demand (type 1) transitions.

cut(). Method in class NoFlyZoneContainer

Method Cut removes the selected NoFlyZone

cycle(ValueObj, double, int, int, int, double, int, int, PrintFlag). Static method in class TabuMod

cycle - updates the search parameters if the incumbent tour is found in the hashing structure

CycleOut(). Constructor for class CycleOut

Method findXYN finds the NoFlyZone node (of classTimwWindow) in the NoFlyZone wthout setting current

firstHashVal(int). Static method in class HashMod

firstHashVal

Frame1(). Constructor for class Frame1

Method Frame1 is the constructor

Frame1(String). Constructor for class Frame1

Method Frame1 is the constructor which takes a title as a string

---

G

getArr(). Method in class NodeType

getDep(). Method in class NodeType

GetDist(). Constructor for class GetDist

getDistanceBetween(LatLong, LatLong). Method in class ConformalMap

Method getDistanceBetween returns the great circle distance between 2 points

getEa(). Method in class NodeType

getId(). Method in class NodeType

getLa(). Method in class NodeType

getLat(). Method in class LatLong

Method getLat returns the Lattitude as a Double

getLatDegrees(). Method in class LatLong

Method getLatDegrees returns the Degrees part of the Lattitude as an Integer

getLatDegrees(). Method in class NodeType

getLatMinutes(). Method in class LatLong

Method getLatDegrees returns the Minutes part of the Lattitude as an Integer

getLatMinutes(). Method in class NodeType

getLatSeconds(). Method in class LatLong

Method getLatDegrees returns the Seconds part of the Lattitude as an Integer

getLatSeconds(). Method in class NodeType

getLoad(). Method in class NodeType

getLocked(). Method in class NodeType

getLong(). Method in class LatLong

Method getLong returns the Longitude as a Double

getLongDegrees(). Method in class LatLong

Method getLatDegrees returns the Degrees part of the Longitude as an Integer

getLongDegrees(). Method in class NodeType

getLongMinutes(). Method in class LatLong

Method getLatDegrees returns the Minutes part of the Longitude as an Integer

getLongMinutes(). Method in class NodeType

getLongSeconds(). Method in class LatLong

Method getLatDegrees returns the Seconds part of the Longitude as an Integer

getLongSeconds(). Method in class NodeType

getM(). Method in class NodeType

getNode(). Method in class Target

Method getNode returns the node

getNumberOfVehicles(). Method in class AirCraftCharacteristics

Method getNumberOfVehicles returns the number of the UAVs

getQty(). Method in class NodeType

getRange(). Method in class AirCraftCharacteristics

Method getRange returns the range of the UAV

getSpeed(). Method in class AirCraftCharacteristics

setLatDegrees(int). Method in class LatLong

Method setLatDegrees sets theDegrees part of the Lattitude using an Integer

setLatMinutes(int). Method in class LatLong

Method setLatMinutes sets the Minutes part of the Lattitude using an Integer

setLatSeconds(int). Method in class LatLong

Method setLatSeconds sets the Seconds part of the Lattitude using an Integer

setLoad(int). Method in class NodeType

setLong(double). Method in class LatLong

Method setLong sets the Longitude using a Double

setLongDegrees(int). Method in class LatLong

Method setLongDegrees sets the Degrees part of the Longitude using an Integer

setLongMinutes(int). Method in class LatLong

Method setLongMinutes sets the Minutes part of the Longitude using an Integer

setLongSeconds(int). Method in class LatLong

Method setLatMinutes sets the Seconds part of the Longitude using an Integer

setNextTarget(Target). Method in class Target

Method setNextTarget sets the next Target

setNode(NodeType). Method in class Target

Method setNode sets the current node

setNumberOfVehicles(int). Method in class AirCraftCharacteristics

Method setNumberOfVehicles sets the number of UAVs

setPreviousTarget(Target). Method in class Target

Method setPreviousTarget sets the previous Target

setQty(int). Method in class NodeType

setRange(double). Method in class AirCraftCharacteristics

Method setRange sets the range of the UAV

setSpeed(double). Method in class AirCraftCharacteristics

Method setSpeed sets the speed of the UAV

setThreat(TimeWindow). Method in class ThreatCharacteristicsDialog

Method setThreat sets the threat you are editing as a TimeWindow

setTimeWindow(TimeWindow). Method in class TargetCharacterisitcsWindow

Method setTimeWindow sets the TimeWindow

setType(int). Method in class NodeType

setVisible(boolean). Method in class AboutDialog

Method setVisible shows or hides the About Dialog Box

setVisible(boolean). Method in class AircraftCharacteristicsF

Shows or hides the component depending on the boolean flag b.

setVisible(boolean). Method in class QuitDialog

Shows or hides the component depending on the boolean flag b.

setVisible(boolean). Method in class TargetListFrame

Shows or hides the component depending on the boolean flag b.

setVisible(boolean). Method in class ThreatCharacteristicsDialog

Shows or hides the component depending on the boolean flag b.

setWait(int). Method in class NodeType

setX(int). Method in class NodeType

setX(int). Method in class Target

Method setX sets the x coordinate

setX(int). Method in class Xy

Method setX sets the X coordinate

setY(int). Method in class NodeType

setY(int). Method in class Target

Method setY sets the Y coordinate

TargetListFrame(String). Constructor for class TargetListFrame

TargetListFrame(Target). Constructor for class TargetListFrame

Method TargetListFrame is a constructor taking a Target

ThreatCharacteristicsDialog(TimeWindow). Constructor for class ThreatCharacteristicsDialog

Method ThreatCharacteristicsDialog is the constructor

TimeMatrix(). Constructor for class TimeMatrix

timeMatrix(int, int, double, int, CoordType[], int[]). Static method in class TimeMatrixObj

Compute 2 dimensional time/distance matrix Does not assume the problem is symmetric, but makes it so

TimeMatrixObj(). Constructor for class TimeMatrixObj

timePrint. Variable in class PrintFlag

Timer(). Constructor for class Timer

toString(). Method in class KeyObj

toString(). Method in class RecordObj

toString(). Method in class ValueObj

totalSeconds. Variable in class Timer

totalSeconds(). Method in class Timer

totPenalty. Variable in class SearchOut

totPenalty. Variable in class StartPenBestOut

totPenalty. Variable in class TsptwPenOut

tour. Variable in class SearchOut

tourCost. Variable in class SearchOut

tourCost. Variable in class StartPenBestOut

tourCost. Variable in class TsptwPenOut

tourHVwz(NodeType[], int[]). Static method in class HashMod

Appendix 2. Class Hierarchy

- class java.lang.Object
    - class AirCraftCharacteristics
    - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
        - class java.awt.Container
            - class java.awt.Panel
                - class myToolbarTestPanel (implements java.awt.event.ActionListener)
            - class java.awt.ScrollPane
                - class myScrollPane
            - class java.awt.Window
                - class java.awt.Dialog
                    - class AboutDialog
                    - class QuitDialog
                - class java.awt.Frame (implements java.awt.MenuContainer)
                    - class AircraftCharacteristicsF
                    - class Frame1
                    - class TargetCharacterisitcsWindow
                    - class TargetListFrame
                    - class ThreatCharacteristicsDialog
    - class ConformalMap
    - class CoordType
    - class CycleOut
    - class GetDist
    - class HashMod
    - class InFromKeybd
    - class KeyObj
    - class KeyToString
    - class KeyboardTest
    - class LatLong
    - class MTSPTW
        - class BestSolnMod
        - class TsptwPen
    - class NoCycleOut
    - class NoFlyZoneContainer
    - class NodeType

- class PrintCalls
- class PrintFlag
- class ReacTabuObj
- class ReadFile
- class RecordObj
- class SearchOut
- class StartPenBestOut
- class StartTourObj
- class TabuMod
- class Target
- class TimeMatrix
- class TimeMatrixObj
- class Timer
- class TsptwPenOut
- class TwBestTTOut
- class ValueObj
- class VrpPenType
- class WriteFile
- class Xy

Bibliography

Allison, David. "MAPPRO: A Program For Processing The Projection of Latitude-Longitude Coordinates Into Rectangular Map Coordinate Systems," *Computers & Geosciences, 21*: 859-875 (1995)

Angehrn, Albert A. and Lüthi Hans-Jakob "Intelligent Decision Support Systems: A Visual Interactive Approach," *Interfaces 20*: 17-28 (November-December 1990)

Angehrn, Albert A. "Modeling by Example: A link between users, models, and methods in DSS," *European Journal of Operational Research, 55*: 296-308 (1991)

Basnet, Chuda, Les Foulds, and Magid Igbaria. "FleetManager: a microcomputer-based decision support for vehicle routing," *Decision Support Systems 16*: 195-207 (1996)

Bertsimas, Dimitris J. and David Simchi-Levi. "A New Generation of Vehicle Routing Research: Robust Algortihms, Addresing Uncertainty," Operations *Research, 44*: 286-304 (March-April 1996)

Bortoluzzi,Giovanni and Marco Ligi, "DIGMAP: A Computer Program For Accurate Acquisition By Digitizer Of Geographical Coordinates From Conformal Projections," *Computers & Geosciences 12*: 175-197 (1986)

Crossland, M. D., B. E. Wynne, and W. C. Perkins. "Spatial decision support systems: An overview of technology and a test of effacy," *Decision Support Systems 14*: 219-235 (1995)

Dana, Peter H. "The Geographer's Craft Project, Department of Geography, The University of Texas at Austin," Map Projections explained, n pag. http://wwwhost.cc.utexas.edu/ftp/pub/grg/gcraft/notes/mapproj/mapproj.html (1995)

Gendreu, Michel, Gilbert Laporte, and René Séguin. "Atabu Search Heuristic For The Vehicle Routing Problem With Stochastic Demands And Customers," *Operations Research, 44*: 469-477 (May-June 1996)

Hill, Greg. "Should they know the Truth About Map Projections?" *Queensland Geographical Journal, 4*: 47-60 (1989)

Holsapple C. W. S. Park, and A. B. Whinston. "Framework for DSS Interface Development" in *Recent Developments in Decision Support Systems, NATO ASI Series* . Ed. Clyde W. Holsapple and Andrew Whinston. Berlin: Springer-Verlag (1991)

Jones, Christopher V. "User Interface Development and Decision Support Systems" in *Recent Developments in Decision Support Systems, NATO ASI Series* . Ed. Clyde W. Holsapple and Andrew Whinston. Berlin: Springer-Verlag, 1991

Keenan, Peter B. "Spatial decision support systems for vehicle routing," *Decision Support Systems, 22*: 65-71 (January 1998)

O'Rourke, Kevin P. *Dynamic Unmanned Aerial Vehicle (UAV) Routing With a Java-encoded Reactive Tabu Search Metaheuristic.* MS thesis, AFIT/GOA/ENS/99M-06. School of Operations Research, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1999

Ryan, J. L., T.G. Bailey, J.T. Moore, and W. B. Carlton. "Unmanned Aerial Vehicle (UAV) Route Selection using Reactive Tabu Search," to appear in *Military Operations Researc,* (1999)

Sisson, Mark. *Applying Tabu Heuristic To Wind Influenced MinimumRisk and Maximum Expected Coverage Routes.* MS thesis, AFIT/GOR/ENS/97M-20. School of Operations Research, Air Force Institute of Technology (AU), Wright-Patterson AFB OH (March 1997)

Taylor, Alan D. "Conformal Map Transformations for Meteorological Modelers," *Computers and Geosciences, 23*: 63-75 (1997)

Theisen, Paul, "USAF Unmanned Aerial Vehicle Battlelab", UAV Battlelab homepage, n. pag. http://www.wg53.eglin.af.mil/battlelab/default.html (2 March 1999)

Walston, Jennifer G. *Unmanned Aerial Vehicle Engagement Level Simulation.* MS thesis, AFIT/GOR/ENS/99M-17. School of Operations Research, Air Force Institute of Technology(AU) , Wright-Patterson AFB OH (March 1999)

Vita

1Lt Randy Flood was born on 3 November 1972 in Coquille, Oregon. He graduated from North Bend High School in North Bend, Oregon in June 1991. He entered undergraduate studies at Oregon State University in Corvallis, Oregon where he graduated with a Bachelor of Science degree in Computer Science in June 1995. He was commissioned through the Detachment 685 AFRTOC at Oregon State University.

His first assignment was at Langley AFB as a software engineer in July 1995. While there, he entered the Graduate Computer Science program, at Old Dominion University. In August 1998, he transferred to the Air Force Institute of Technology, and entered the Graduate Computer Science program, School of Engineering, Air Force Institute of Technology. Upon graduation, he will be assigned to the Air Force Communications Agency.

Permanent Address: 141 N. Gould
Coquille, OR 97423

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March, 1999	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE A JAVA BASED HUMAN COMPUTER INTERFACE FOR A UAV DECISION SUPPORT TOOL USING CONFORMAL MAPPING			**5. FUNDING NUMBERS**
6. AUTHOR(S) Randy A. Flood, 1LT, USAF			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, 2950 P Street, WPAFB OH 45433-7765			**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCS/ENS/99M-1
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) O'HAIR, MARK A., LT COL, USAF UAV Battleab 1003 Nomad Way, Suite 107 Eglin AFB FL 32542-6867			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES T. Glenn Bailey, Lieutenant Colonel, USAF			
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 words)*

This paper describes the development of the Human Computer Interface (HCI) for a Decision Support System for routing Unmanned Aerial Vehicles (UAVs). This problem is a multi-vehicle routing problem with time-windows. Because of the unique nature of UAVs, a tool is needed to support dynamic re-routing. We solve the problem in two ways. First, we create a UAV Decision Support Tool (UAV DST) that uses a set of Java software objects to display maps and convert between latitude-longitude coordinates and x-y coordinates. Secondly, this library provides the ability for the user to dynamically re-optimize large UAV routing problems through a simple graphical interface. The library is built on top of a Java implementation of the tabu search algorithm written by O'Rourke (1999). This library provides the basis for future simulation and analysis of the Kenney Battlelab Initiatives by providing the interface to routing decision support and simulation modules.

14. SUBJECT TERMS UAV, Routing, Decision Support System, TSP			**15. NUMBER OF PAGES** 66
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL